An exploration and practice of an optimized online annotation solution based on PDF slicing and SVG

Zhenduo Li

Research and Development Center, Agricultural Bank of China

zhenduoli@163.com

Abstract. With the continuous development of network technology, online annotation has become an essential tool in fields such as education, training, and document collaboration. However, traditional annotation techniques are plagued by several limitations, including low security, non-erasable annotations after uploading, and difficulties in communication across different systems. To address these issues, this paper proposes an SVG-based online annotation technique that is built upon PDF file slicing. This approach aims to enhance the deficiencies of traditional annotation techniques and improve user interaction. The research begins by introducing the background and significance of online annotation technology, elucidating the application prospects of PDF slicing technology and SVG in annotation. The paper then provides a detailed description of the annotation algorithm and the implementation process. Additionally, it discusses the experimental setup, results, and analysis. The results indicate that the SVG-based online annotation technology can effectively enhance annotation efficiency and real-time interaction while enabling erasure of historical data and facilitating cross-platform communication. Compared to traditional annotation techniques, the SVG-based approach exhibits higher practicality and flexibility, making it suitable for various types of document annotations and collaborative scenarios.

Keywords: file slicing, PNG, SVG, online annotation.

1. Introduction

As network and multimedia technologies rapidly advance, electronic documents have become a popular feature in both work and daily life. These documents are easily created, edited, accessed, and modified at any time and from any location. They facilitate rapid transmission and sharing over the network, enabling efficient collaboration and communication among different departments and individuals, thus overcoming the slow and inefficient transfer issues associated with traditional paper documents. However, the inability to annotate electronic documents as easily as their paper counterparts has emerged as a major concern. Users are in dire need of an online annotation function that allows them to add their own comments, annotations, or explanations to electronic documents, facilitating better document reading, discussion, and research. Although some whiteboard note-taking tools in the application market currently enable online document annotation, they typically synthesize files and annotation strokes into image formats such as PNG or JPG for transmission. Moreover, the data format of annotations between IOS and Android systems cannot be unified, rendering users unable to modify and erase previous content after uploading annotations. These issues constrain the efficiency and user experience of annotations. Therefore, this study aims to explore an SVG-based online annotation solution based on PDF file slicing to address these challenges. This technology first slices PDF files, independently stores different page elements, and generates annotation drawing boards. Simultaneously, it employs Scalable Vector Graphics (SVG) technology to realize dynamic interaction and real-time synchronization of annotation content, thus improving the efficiency and experience of online annotation.

2. Related Work

Online annotation technology has gradually matured. Using technologies such as touch screens and styluses on mobile devices, many mobile applications have implemented online annotation functions on documents, which have been widely applied. Research has shown that using online annotation tools is more conducive to rapid knowledge acquisition [1], and most respondents believe that online annotation tools are more convenient and faster than handwritten paper notes [2]. In the field of education, some mobile applications enable students and teachers to annotate and comment on electronic documents, facilitating anytime, anywhere learning and communication [3,4]. In the office environment, some mobile applications provide document annotation and review functions, enabling office workers to modify and review documents on mobile devices at any time, thus achieving efficient collaboration and communication [5]. In the field of project management, some mobile applications offer document annotation and tracking functions, allowing project managers to annotate and track project documents anytime, anywhere, thus improving the efficiency and accuracy of project management [6,7].

3. Specific Implementation of Online Annotation Solution

Portable Network Graphics (PNG) is a bitmap format that uses lossless compression algorithms. It supports index, grayscale, RGB color schemes, and features such as the Alpha channel. Its design purpose is to replace the GIF and TIFF file formats while adding some features that the GIF format does not possess. PNG uses a lossless data compression algorithm derived from LZ77. The extension of PNG files is .png.

Scalable Vector Graphics (SVG) is an open standard for drawing vector graphics. SVG can store various types of graphics, including lines, shapes, and text, and supports multiple color modes and transparency. In addition, SVG supports animation effects such as scaling, rotation, and deformation. The main features of SVG include scalability, supporting multiple fonts and styles, and running on different operating systems and platforms; high compatibility, compatible with most browsers, including IE, Chrome, and other mainstream browsers; high quality, providing clear and sharp images with high-quality vector graphics; easy maintenance, objects can be controlled through CSS or JavaScript, facilitating modification and updates; and cross-platform capability, running on different platforms such as Windows, MacOS, and Linux without the need for additional plug-ins. In summary, SVG technology is an efficient, flexible, and scalable vector graphics drawing technology widely used in web design, printing, and publishing.

The annotation of electronic documents is often presented in the form of lines, and the background of annotations is mostly white, which is well suited for SVG's file description approach. However, PNG requires a large number of white pixels to fill the image, leading to substantial storage space consumption. Therefore, using SVG as the unified format for annotation data transmission will provide a better user experience. SVG uses XML format to define graphics and has some predefined element tags that developers can directly use and manipulate, such as the rectangle <rect>, circle <circle>, ellipse <ellipse>, line <line>, polyline <polyline>, polygon <polygon>, and path <path>. Figure 1 shows the image drawn using SVG-defined tags, and the text box below presents the corresponding SVG text for Figure 1.

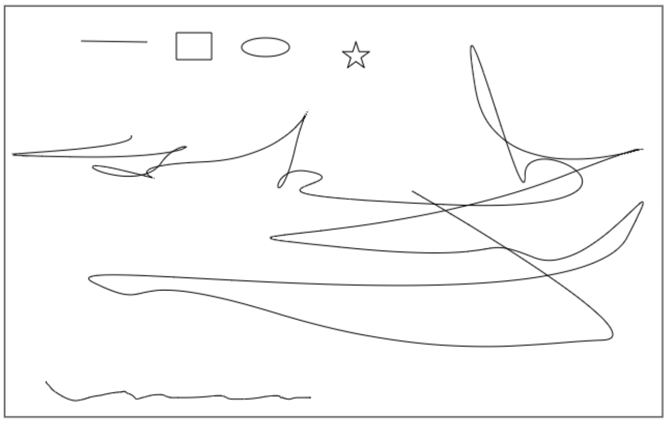

Figure 1. SVG Path Diagram

The XML text corresponding to Figure 1 is as follows:

```
        <svg width="800" height="600" xmlns="http://www.w3.org/2000/svg">

         <g>
          <title>Layer 1</title>
          <line stroke-linecap="undefined" stroke-linejoin="undefined" id="svg_1" y2="64" x2="165" y1="63" x1="101"
stroke="#000" fill="none"/>
          <rect id="svg_2" height="26" width="34" y="55" x="193" stroke="#000" fill="#fff"/>
          <ellipse ry="9" rx="23" id="svg_3" cy="69" cx="279" stroke="#000" fill="#fff"/>
          <path id="svg_4" d="m353.23,73.27142l9.93119,0l3.06881,-9.54918l3.06881,9.54918l9.93118,0l-
8.03448,5.90165l3.06897,9.54918l-8.03449,-5.90181l-8.03449,5.90181l3.06897,-9.54918l-8.03449,-5.90165z" stroke="#000"
fill="#fff"/>
          <path id="svg_12" d="m149.28572,153.71429c11,18 -212,18 -59,21c153,3 111,-27 88,6c-23,33 -110,-10 -38,7c72,17 -
31,-3 78,-8c109,-5 107,-87 89,-12c-18,75 -31,10 17,22c48,12 -120,20 126,31c246,11 79,-79 79,-28c0,51 -70,-225 -48,-
85c22,140 273,12 89,84c-184,72 -396,52 -221,70c175,18 128,-14 188,10c60,24 139,-114 91,-20c-48,94 -580,13 -514,45c66,32
14,-20 175,29c161,49 250,28 314,26c64,-2 -183,-144 -183,-144" opacity="NaN" stroke="#000" fill="none"/>
          <rect stroke="#000" id="svg_13" height="396.7143" width="634.71431" y="30.57143" x="27.28568" fill="none"/>
          <path d="m67.14287,391.42859c0,1.42856 2.41793,3.12286 4.28571,5.71426c2.36259,3.27792 6.9736,5.98611
11.42857,8.57144c5.52571,3.2067 11.46546,4.74338 14.28571,4.28571c4.45921,-0.72363 7.14287,-1.42856 11.42857,-
2.85712c4.28572,-1.42859 9.90139,-1.76819 14.28571,-2.85715c8.43343,-2.09467 14.11222,-2.13351 18.57143,-
2.85715c2.82025,-0.45767 2.4192,0.65543 4.28571,1.42856c1.31982,0.54669 4.28572,1.42859
5.71429,2.85715c1.42859,1.42859 2.85715,2.85715 2.85715,2.85715c1.42857,0 5.71428,-1.42856 11.42857,-2.85715c5.71428,-
1.42856 8.57143,-1.42856 11.42857,-1.42856c2.85713,0 3.84776,0.65543 5.71428,1.42856c2.63966,1.09338 4.28572,1.42859
7.14285,1.42859c2.85715,0 7.14287,0 12.85715,0c8.57143,0 17.03687,-0.71082 24.28572,-1.42859c7.10809,-0.70386 10,0
11.42857,0c0,0 0.99062,0.65546 2.85715,1.42859c1.31981,0.54669 2.85713,1.42856 5.71428,1.42856c2.85715,0 5.71429,0
11.42856,0c8.57144,0 17.03687,-2.13937 24.28574,-2.85715c5.68649,-0.56308 8.57141,0 10,0c0,0 0,1.42859
1.42856,1.42859c1.42859,0 2.60538,0.69464 5.71429,1.42856c2.78073,0.65643 5.71429,-1.42856 10,-1.42856c5.71429,0
8.57144,0 10,0c1.42859,0 1.42859,0 1.42859,0l11.42856,0l0,0" id="svg_14" stroke="#000" fill="none"/>
         </g>
        </svg>
```

In the proposed solution in this paper, the main SVG element used is the path <path>. The path can be displayed in multiple ways, including M = moveto, L = lineto, H = horizontal lineto, V = vertical lineto, C = curveto, S = smooth curveto, Q = quadratic Bézier curve, T = smooth quadratic Bézier curveto, A = elliptical Arc, Z = closepath. Therefore, the urgent technical issue to be addressed is the mutual conversion between the Android and iOS native drawing board trajectories and SVG <path>. This conversion, in conjunction with file slicing technology, generates the corresponding SVG file.

3.1. Server-Side File Slicing

The technique of dividing a PDF file into one or more PNG format images is referred to as file slicing. By using file slicing technology, the uploaded PDF file is divided into PNG images and saved on the server. The mobile end can directly access the file slices through the URL without downloading the original PDF file to the local device, thereby meeting security requirements and facilitating file previews.

The slicing service downloads the original PDF file and uses a thread pool to process each page of the file concurrently, ensuring efficient conversion. Additionally, it ensures the quality of the file without exceeding the file storage limit. The following example code illustrates the process:

```
    /**
       * PDF asynchronous slicing in online annotations
       * @param filePath
       * @param fileId
       * @param pictureSaveDirPath    */
     public void asyncTaskOfComment(Path filePath, String fileId, Path pictureSaveDirPath) {
            long startTime = System.currentTimeMillis();
            THREAD_POOL_SPLIT.execute(()-> {
                   try {
                          long startThreadTime = System.currentTimeMillis();
                          int pages = PdfToImage2.convert2JPG(filePath, pictureSaveDirPath);
                          CommentEntity commentEntity = new CommentEntity();
                          commentEntity.setId(fileId);
                          commentEntity.setPages(String.valueOf(pages));
                          commentDAO.updateByKey(commentEntity);
                          int workQueue = ((Queue) FieldUtils.getField(ThreadPoolExecutor.class, "workQueue",
true).get(THREAD_POOL_SPLIT)).size();
                   } catch (Exception e) {
                          LOG.error("", e);
                   }
            });
     }
```

3.2. Conversion of Annotation Trajectories into SVG Paths

Firstly, it is necessary to obtain the server's file slices of the PDF file through the slicing interface. The slices are specifically in the form of PNG images, which are loaded via URL and implemented using lazy loading to achieve smooth page flipping for annotations. After rendering the annotation background, a drawing board with a 1:1 scale is generated based on the slices. The drawing board is a layer positioned above the sliced background. Users can select the color, thickness, and opacity of the brush and then draw corresponding trajectories on the screen using their fingers or a touch pen. Each time a stroke is initiated and lifted on the drawing board, a trajectory is generated. The native code converts each of these trajectories into SVG paths <path/>. This process is elaborated for Android and iOS through specific code logic combined with annotations.

Conversion of Android Canvas Drawing Board Trajectories to SVG

The following is an example code:

```
switch (action) {
  //After listening to the drawn action, generate the initial coordinates of the trajectory and initialize the trajectory //object
      case MotionEvent.ACTION_DOWN:
            pathPoint = "";
            mLastX = x;
            mLastY = y;
        //M means Move the starting point coordinates of the trajectory to (x, y)

            pathPoint = "M" + x + " " + y;
            if (mPath == null) {
                  mPath = new Path();
            }
            mPath.moveTo(x, y);
            break;
//Monitor the sliding operation of the brush on the screen, calculate the coordinates of each landing point through //the
Bezier curve formula, and concatenate them into the pathPoint string
      case MotionEvent.ACTION_MOVE:
      //The purpose of setting the endpoint as the center point of two points here is to make the drawn curve //smoother. If
      the endpoint is directly set to x, y, the effect is the same as lineto, but it is actually a polyline //effect
            mPath.quadTo(mLastX, mLastY, (x + mLastX) / 2, (y + mLastY) / 2);
            if (mBufferBitmap == null) {
                  initBuffer();
            }
            if (mMode == Mode.ERASER && !mCanEraser) {
                  break;
            }
```

```
                //The drawPath method provided by the Canvas palette will draw trajectory images based on coordinates
                 mBufferCanvas.drawPath(mPath, mPaint);
                //L means each point is connected by a straight line
                 pathPoint = pathPoint + "L" + x + " " + y;
                 invalidate();
                 mLastX = x;
                 mLastY = y;
                 break;
    //After listening to the brush contacts leaving the screen, calculate the transparency, thickness stroke width, color //stroke,
  and other related attributes of the brush, and combine them with pathPoint to fully concatenate the SVG //label of the current
  trajectory<path..../>
           case MotionEvent.ACTION_UP:
                 String hexColor = "#" + Integer.toHexString(mPaint.getColor()).substring(2);
                 float opacity = (float)mPenAlpha / 255;
                 //This is a complete SVG path
                 pathPoint = "<path d='" + pathPoint + "'" +
                             " stroke='" + hexColor + "'" + " " + "stroke-width='" +
                             mPaint.getStrokeWidth() + "'" + " " + "fill='none'" + " " + "opacity='" + opacity + "'" + "/>";
                 if (newAddedSvgPaths == null){
                       newAddedSvgPaths = new ArrayList<>();
                 }
                 //Add path to path array
                 newAddedSvgPaths.add(pathPoint);
                 if (mLastSvgPaths == null){
                       mLastSvgPaths = new ArrayList<>();
                 }
                 mLastSvgPaths.add(pathPoint);
                 if (mMode == Mode.DRAW || mCanEraser) {
                       saveDrawingPath();
                 }
                 mPath.reset();
                 break;
     }
```

iOS Processing Logic

SVG File Generation: This section primarily involves saving the curve data from the drawing board and generating content that complies with the SVG standard. This process includes merging old SVG content and handling eraser path data. An example code snippet is provided below:

```
    // Previous SVG rendering model
           SVGRenderer *oldSvgRender = [[SVGRenderer alloc] initWithString:oldSVG];
           CGRect oldSVGRect = oldSvgRender.viewRect; //Previous SVG view viewBox rectangle
           // SVG header
           NSMutableArray *svgStringList = [NSMutableArray new];
           [svgStringList addObject:@"<?xml version='1.0' encoding='UTF-8' standalone='no'?>"];
           [svgStringList          addObject:@"<svg          version='1.1'          xmlns='http://www.w3.org/2000/svg'
 xmlns:xlink='http://www.w3.org/1999/xlink' "];
           [svgStringList addObject:[NSString stringWithFormat:@"width='%.1f' height='%.1f' viewBox='0, 0, %.1f, %.1f' >",

                                                   oldSVGRect.size.width,
                                                   oldSVGRect.size.height,oldSVGRect.size.width,oldSVGRect.size.height]];
           // Determine if it is necessary to inject previous SVG content
           if (oldSVG && oldSVG.length > 0)
           {
                 [svgStringList addObject:[self queryContentFromSVG:oldSVG]];
           }
           // Eraser Path
           [svgStringList addObject:@"<defs>"];
           // Based on the old SVG view size, perform matrix scaling on the new path
           CGFloat              scaleX = (oldSVGRect.size.width / viewBoxSize.width);
           CGFloat              scaleY = (oldSVGRect.size.height / viewBoxSize.height);
           CGAffineTransform trans    = CGAffineTransformMakeScale(scaleX, scaleY);
           NSMutableArray<PathInfo *> *pathListCopy = [NSMutableArray new];

           // Copy a new data and perform matrix transformation on the curve data
           for (PathInfo *pathInfo in pathList)
           {
                 PathInfo *pathInfoCopy = [pathInfo clone];
                 [pathInfoCopy.path applyTransform:trans];
                 pathInfoCopy.path.lineWidth = (pathInfo.brush.width / UIScreen.mainScreen.scale) * scaleY;
                 [pathListCopy addObject:pathInfoCopy];
```

```
        }
        // Generate SVG content
        for (NSUInteger i = 0; i < pathListCopy.count; i += 1) {
            PathInfo *pathInfo = pathListCopy[i];
            pathInfo.tag = [NSString stringWithFormat:@"path%lu", i];
            if ([pathInfo.brush.color isEqual:[UIColor clearColor]]) {
                [svgStringList addObject:[NSString stringWithFormat:@"<mask id='%@'>", pathInfo.tag]];
                [svgStringList addObject:[NSString stringWithFormat:@"<path fill='#FFF' d='M0 0h%.1fv%.1fH0z'/>",
oldSVGRect.size.width, oldSVGRect.size.height]];
                [svgStringList addObject:[NSString stringWithFormat:@"<path d='%@' stroke='#000' stroke-
width='%.1f' fill='none' stroke-linecap='round' />",
                                                    [SVGPathGenerator svgPathFromCGPath:pathInfo.path.CGPath],
                                                    pathInfo.path.lineWidth]];
                [svgStringList addObject:@"</mask>"];
            }
        }
        [svgStringList addObject:@"</defs>"];
        // Trajectory path
        [self buildPathArray:[pathListCopy mutableCopy] withSource:svgStringList];
        [svgStringList addObject:@"</svg>"];
```

3.3. Rendering SVG Paths to the Annotation Drawing Board

Implementation Logic on Android

The Android implementation logic involves parsing the SVG string data corresponding to each page's file slice returned by the Document. This data is combined with the current drawing board's width and height, as well as the SVG's ViewBox dimensions, to calculate the ratio. This facilitates the transformation of the trajectory coordinates on the drawing board. The following is an example code snippet:

```
//Convert SVG string to Document object
Document document = Jsoup.parse(svgStr);
Elements elements = document.select("path");
Elements elementsvgs = document.select("svg");
float wScale = 1;
for(Element element : elementsvgs){
    //Perform matrix transformation on SVG data annotated on each page
        String viewBox = element.attr("viewBox");
        float svgW = Float.parseFloat(viewBox.split(", ")[2]);
        float svgH = Float.parseFloat(viewBox.split(", ")[3]);
        wScale = w / svgW;
        hScale = h / svgH;
        break;
  }
```

The <d/> tag is used to retrieve the attributes of each trajectory, including color, thickness, opacity, etc. These attributes are then converted into Path objects that the Canvas drawing board can recognize. The following example code snippet demonstrates this process:

```
    //Each element represents an SVG path
    for (Element element : elements) {
    Path mPath = new Path();
    Paint mPaint = new Paint();
    String path = element.attr("d");
    if (wScale == 1){
          paths.add(String.valueOf(element));
    }
   //Obtain the thickness, color, transparency, and other attributes of the brush
    String stroke = element.attr("stroke");
    String stroke_width = element.attr("stroke-width");
    String endStr = "' stroke='" + stroke + "' stroke-width='" + stroke_width + "'";
    float opacity = 1;
    if (element.hasAttr("opacity")){
          opacity = Float.parseFloat(element.attr("opacity"));
          endStr = endStr + " opacity='" + opacity + "'";
    }
    endStr = endStr + ">";
    try{
       //Convert to Path Object
          parser(paths, endStr, path, mPath, wScale, hScale);
    }catch (Exception e){
          e.printStackTrace();
          Toast.makeText(mContext, e.getMessage(), Toast.LENGTH_LONG);
    }
    mPaint.setColor(Color.parseColor(stroke));
    mPaint.setAlpha((int) (opacity * 255));
    mPaint.setStrokeWidth(Float.parseFloat(stroke_width));
    Map<Paint, Path> map = new HashMap<>();
    map.put(mPaint, mPath);
    pathList.add(map);
}
```

This code exemplifies the process of rendering SVG paths to the annotation drawing board on the Android platform, detailing the crucial steps involved in the transformation and conversion of SVG data to the corresponding graphical representation on the drawing board.

```
public Path parser(List<String> paths, String endStr, String svgPath, Path lPath, float wScale, float hScale) {
      String path = "<path d='";
      mIndex = 0;
      lPath.setFillType(Path.FillType.WINDING);
      //Record the last operating point
      PointF lastPoint = new PointF();
      findCommand(svgPath);
      for (int i = 0; i < cmdPositions.size() - 1; i++) {
            Integer index = cmdPositions.get(i);
         //Connect the points of each coordinate according to commands such as M, L, H, V, C, Q, S, T, etc
            switch (svgPath.charAt(index)) {
                  case 'm':
                  case 'M': {
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(ps[0], ps[1]);
                        lPath.moveTo(lastPoint.x, lastPoint.y);
                        path = path.concat("M" + ps[0] + " " + ps[1]);
                  }
                  break;
                  case 'l':
                  case 'L': {
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(ps[0], ps[1]);
                        lPath.lineTo(lastPoint.x, lastPoint.y);
                        path = path.concat("L" + ps[0] + " " + ps[1]);
                  }
                  break;
                  case 'h':
                  case 'H': {
               //Based on the previous coordinate, draw a line horizontally, so the y-axis remains unchanged
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(ps[0], lastPoint.y);
                        lPath.lineTo(lastPoint.x, lastPoint.y);
                        path = path.concat("H" + ps[0]);
                    }
                    break;
                    case 'v':
                    case 'V': {
                    //Based on the previous coordinate, draw a line horizontally, so the x-axis remains unchanged
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(lastPoint.x, ps[0]);
                        lPath.lineTo(lastPoint.x, lastPoint.y);
                        path = path.concat("V" + ps[0]);
                    }
                    break;
                    case 'c':
                    case 'C': {
                    //Cubic Bezier curve
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(ps[4], ps[5]);
                        lPath.cubicTo(ps[0], ps[1], ps[2], ps[3], ps[4], ps[5]);
                        path = path.concat("C" + ps[0] + " " + ps[1] + " " + ps[2] + " " + ps[3] + " " + ps[4] + " " + ps[5]);
                    }
                    break;
                    case 's':
                    case 'S': {
                    //Generally, S will be used after the C or S command, using the previous point as the starting control //point
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lPath.cubicTo(lastPoint.x,lastPoint.y,ps[0], ps[1], ps[2], ps[3]);
                        lastPoint.set(ps[2], ps[3]);
                        path = path.concat("S" + ps[0] + " " + ps[1] + " " + ps[2] + " " + ps[3]);
                    }
                    break;
                    case 'q':
                    case 'Q': {//quadratic Bezier curve
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lastPoint.set(ps[2], ps[3]);
                        lPath.quadTo(ps[0], ps[1], ps[2], ps[3]);
                        path = path.concat("Q" + ps[0] + " " + ps[1] + " " + ps[2] + " " + ps[3]);
                    }
                    break;
                    case 't':
                    case 'T': {//The T command will be used after Q, using the end point of Q as the starting point
                        float ps[] = findPoints(svgPath, i, wScale, hScale);
                        lPath.quadTo(lastPoint.x,lastPoint.y,ps[0], ps[1]);
                        lastPoint.set(ps[0], ps[1]);
                        path = path.concat("T" + ps[0] + " " + ps[1]);
                    }
                    break;
                    case 'a':
                    case 'A':{
                    //Draw an arc
                    }
                    break;
                    case 'z':
                    case 'Z': {
                    //finish
                        lPath.close();
                    }
                    break;
                }
            }
        path = path.concat(endStr);
        if (wScale != 1){
            paths.add(path);
        }
        return lPath;
    }
```

iOS Aspect

The drawing of curves on the iOS drawing board is primarily accomplished within the device's touch event callback function. To achieve smoother line drawing, special handling is applied to the control points of the curves (Bezier curves). Some core code snippets are as follows:

```
open override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
            …
            // Record a coordinate point when the user initially touches the canvas
             let touch = touches.first
             self.points[0] = touch?.location(in: self) ?? CGPoint.zero
            …
    }

    // Bezier curve drawing
     open override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
             …
             // Curve drawing processing
             let touch = touches.first
             let currentPoint = touch?.location(in: self)
             self.pointMoved = true
             self.pointIndex += 1
             self.points[self.pointIndex] = currentPoint ?? CGPoint.zero
              // Bezier curve control point processing
             if self.pointIndex == 4 {
                 self.points[3] = CGPoint(x: (self.points[2].x + self.points[4].x)/2.0, y: (self.points[2].y + self.points[4].y) /
2.0)

                 self.path.move(to: self.points[0])
                 self.path.addCurve(to: self.points[3], controlPoint1: self.points[1], controlPoint2: self.points[2])
                 self.points[0] = self.points[3]
                 self.points[1] = self.points[4]
                 self.pointIndex = 1
             }
            …
        }
       // Merge paths to generate preview images
        open override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
             …
            // Closed Bezier curve
             if !self.pointMoved {      // touchesBegan -> touchesEnded : just touched
                 self.path.move(to: self.points[0])
                 self.path.addLine(to: self.points[0])
                 self.strokePath()
             }
             // Generate preview image
             self.mergePaths()
             self.didUpdateCanvas()
             self.path.removeAllPoints()
             self.pointIndex = 0
             …
        }
```

3.4. Implementation of Eraser Functionality in SVG

Due to the limitations of PNG, it is unable to edit and modify content, thus lacking eraser functionality. SVG supports a feature called mask, where all shapes inside the mask are visible, and those outside the mask are invisible. This feature enables the implementation of the drawing board's eraser function. An example code snippet is provided below:

```
// Sample code for implementing the eraser function using SVG mask
// Hierarchical structure of the eraser path, with different paths covering different areas
// Setting the path color to transparent to achieve the eraser effect
// Storing the data in the SVG's defs tag and accessing it through the mask's URL
svgStringList.append("<defs>")
// Path data
...
```

```
svgStringList.append("</defs>")
// Check if the path is for the eraser by checking its transparency
if pathInfo.brush.color == UIColor.clear {
    // Combining the mask path using the URL method
    source.append("<g mask='url(#\(pathInfo.tag))'>")
    buildPathArray(pathList, withSource: source)
    source.append("</g>")
}
...
```

4. Performance Comparison Analysis of PNG and SVG Annotation Solutions

4.1. Space Occupation Comparison

Since PNG consists of pixels, its size increases with higher resolutions. As annotation data, a large number of uploads may occupy considerable server storage space. In contrast, SVG is essentially a string file composed of tags, occupying minimal space. Taking annotations on the same page of a PDF file as an example, the storage space occupied by uploading the same annotation strokes in PNG and SVG formats to the server is illustrated in Figure 1.

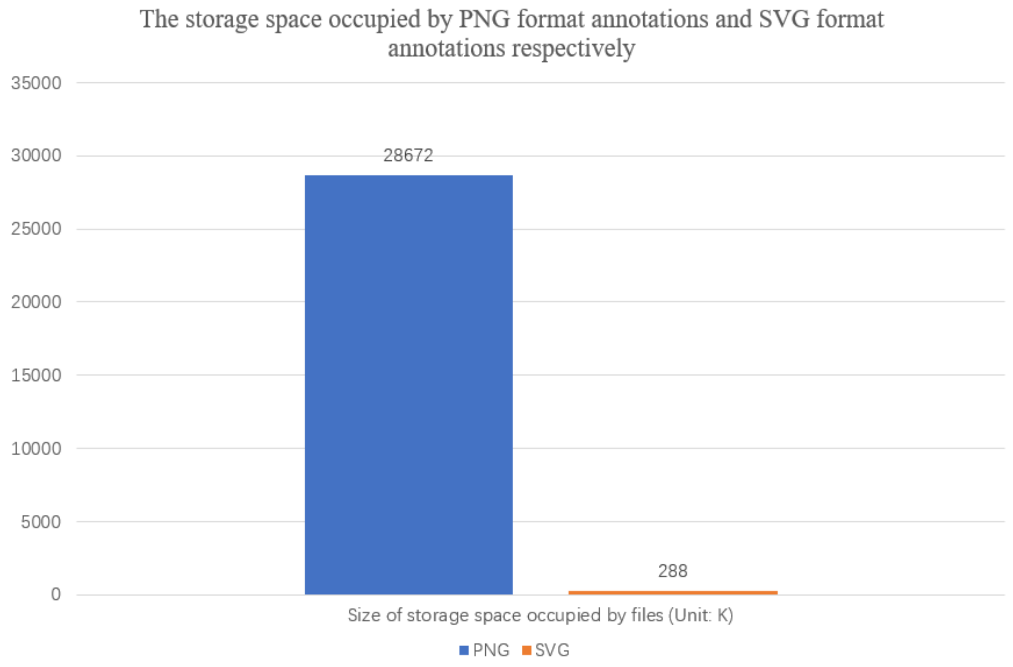

Figure 1. Comparison of Server Storage Space Occupied by PNG and SVG

4.2. Comparison of Upload Speed

Due to the need for data encryption and decryption, the encryption process for PNG files is relatively time-consuming due to their larger storage size, resulting in a longer waiting time for users to upload annotation data after completing their annotations. In contrast, SVG files, being smaller in size, undergo a shorter encryption and decryption process. After conducting several tests, the average time taken to upload annotation trajectory data for the same page of a PDF file in both PNG and SVG formats is as shown in Table 1 below:

Table 1. A Comparison of Time Taken for SVG and PNG Uploads

Data Format	Time (in ms)
PNG	3210
SVG	327

In actual usage, electronic documents often consist of multiple pages. If the PNG method is still employed for online annotations, the file upload time will proportionally increase. In such cases, adopting the SVG method for annotations significantly improves the user experience. Figure 3 illustrates the line graph of the upload time for annotations using both PNG and SVG methods for pages 1 to 6 of a PDF file. It is evident from Figure 3 that the SVG annotation method significantly reduces the file upload time for multi-page document annotations.

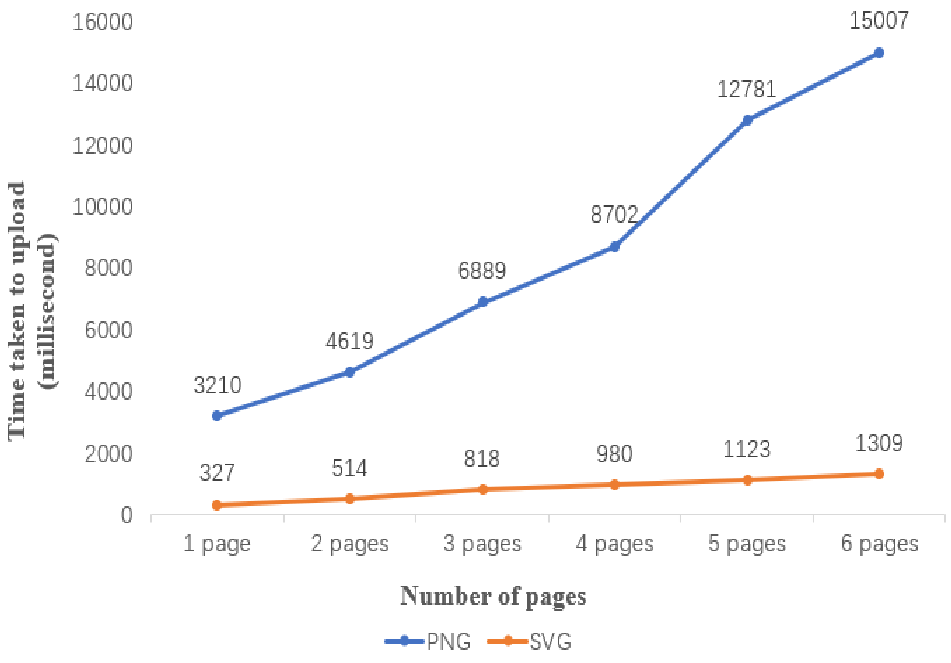

Figure 2. Comparison of Upload Time for Multi-Page Annotations in PNG and SVG (in ms)

4.3. Comparison of Eraser Functionality

The PNG method does not support the erasure of previously annotated trajectories, significantly affecting the user experience. In contrast, the SVG method allows for editing and modification of each annotated trajectory. Using two instances of annotation as an example, the following illustration demonstrates that SVG can erase trajectories from both instances of annotations, while the PNG method can only erase the trajectory from the current instance of annotation.

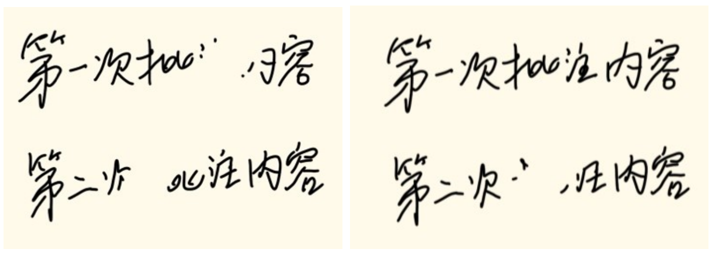

Figure 3. Comparison of Erasure Effects Between PNG and SVG

4.4. Trajectory Effect Comparison

When annotating using the PNG method, each annotation upload involves the synthesis of PNG images. During this process, the PNG images continuously increase in size. To prevent the file from exceeding the storage limit, compression is applied to the PNG file when it becomes too large. This compression gradually reduces the clarity of the annotated trajectories after multiple syntheses. In contrast, the SVG annotation solution separates the file image from the annotation image, ensuring that the image clarity is not compromised by the number of modifications. Taking five annotations as an example, the following illustration depicts the clarity comparison between the two methods:

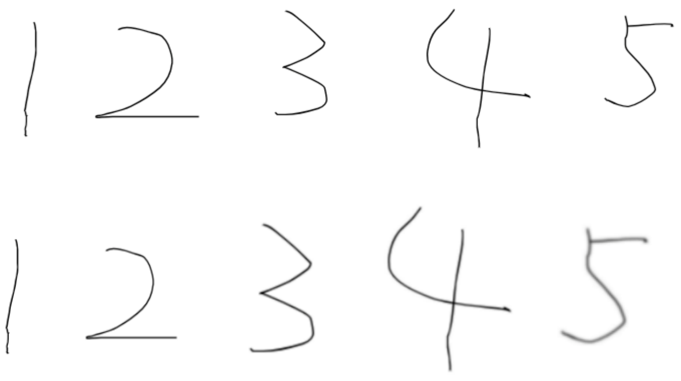

Figure 4. Comparison of Clarity in Annotated Trajectories Between PNG and SVG (SVG on top, PNG on the bottom)

5. Conclusion

The proposed online annotation solution significantly improves the user's upload speed, with this advantage being more pronounced during multi-page annotations. Furthermore, this solution ensures that the clarity of both files and annotations remains at an optimal level, preventing image distortion caused by excessive uploads. Lastly, this solution, bridged by SVG, enables multi-channel annotation data interoperability, establishing a precedent for users to modify their historical annotation data across various platforms. The proposed annotation solution greatly enhances the user experience and holds significant implications for similar tools such as drawing boards and online notepads.

References

[1] Grabe, M., & Christopherson, K. (2005). Evaluating the advantages and disadvantages of providing lecture notes: The role of internet technology as a delivery system and research tool. The internet and higher education, 8(4), 291-298.
[2] Bauer, A., & Koedinger, K. (2005, June). Designing an Online Note Taking Tool from the Ground Up. In EdMedia+ Innovate Learning (pp. 4181-4186). Association for the Advancement of Computing in Education (AACE).
[3] Nor, N. F. M., Azman, H., & Hamat, A. (2013). Investigating Students' Use of Online Annotation Tool in an Online Reading Environment. 3L: Southeast Asian Journal of English Language Studies, 19(3).
[4] Bauer, A., & Koedinger, K. (2005, June). Designing an Online Note Taking Tool from the Ground Up. In EdMedia+ Innovate Learning (pp. 4181-4186). Association for the Advancement of Computing in Education (AACE).
[5] Kieser, A. L., & Golden, F. O. (2009). Using online office applications: Collaboration tools for learning. Distance Learning, 6(1), 41.
[6] Özkan, D., & Mishra, A. (2019). Agile project management tools: a brief comprative view. Cybernetics and Information Technologies, 19(4), 17-25.

[7] Mishra, A., & Mishra, D. (2013). Software project management tools: a brief comparative view. ACM SIGSOFT Software Engineering Notes, 38(3), 1-4.
[8] Boutell, T. (1997). Png (portable network graphics) specification version 1.0 (No. rfc2083).
[9] Eisenberg, J. D., & Bellamy-Royds, A. (2014). SVG essentials: Producing scalable vector graphics with XML. " O'Reilly Media, Inc.".
[10] Bosak, J., & Bray, T. (1999). XML and the second-generation Web. Scientific American, 280(5), 89-93.